Relational Model and Algebra

Introduction to Databases
CompSci 316 Fall 2016
Edgar (Ted) F. Codd (1923-2003)

- Pilot in the Royal Air Force in WW2
- Inventor of the relational model and algebra while at IBM
- The “theoretical foundation” of database management
- A technological revolution!
- Turing Award, 1981
Relational data model

• A database is a collection of relations (or tables)
• Each relation has a set of attributes (or columns)
• Each attribute has a name and a domain (or type)
  • Values are “atomic”
  • Can be strings, integers, reals, characters, ...
  • Cannot be a struct, set, list, array, ...
• Each relation contains a set of tuples (or rows)
  • Each tuple has a value for each attribute of the relation
  • Duplicate tuples are not allowed
    • Two tuples are duplicates if they agree on all attributes

☞ Recall: Simplicity is a virtue!
Example

**User**

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>0.3</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

Ordering of rows doesn’t matter (even though output is always in some order)

“set semantic”

**Group**

<table>
<thead>
<tr>
<th>gid</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>Book Club</td>
</tr>
<tr>
<td>gov</td>
<td>Student Government</td>
</tr>
<tr>
<td>dps</td>
<td>Dead Putting Society</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

**Member**

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>dps</td>
</tr>
<tr>
<td>123</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>abc</td>
</tr>
<tr>
<td>857</td>
<td>gov</td>
</tr>
<tr>
<td>456</td>
<td>abc</td>
</tr>
<tr>
<td>456</td>
<td>gov</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
Schema vs. instance

• **Schema (metadata)**
  • Specifies how the logical structure of data
  • Is defined at setup time
  • Rarely changes
  • But columns can be added/deleted

• **Instance**
  • Represents the data content
  • Changes rapidly, but always conforms to the schema

☞ Compare to **types vs. collections of objects of these types** in a programming language
Example

list of attributes is a set too but a standard order is assumed

• Schema
  • User (uid int, name string, age int, pop float)
  • Group (gid string, name string)
  • Member (uid int, gid string)

• Instance
  • User: \{\langle 142, \text{Bart}, 10, 0.9 \rangle, \langle 857, \text{Milhouse}, 10, 0.2 \rangle, \ldots \}\}
  • Group: \{\langle \text{abc}, \text{Book Club} \rangle, \langle \text{gov}, \text{Student Government} \rangle, \ldots \}\}
  • Member: \{\langle 142, \text{dps} \rangle, \langle 123, \text{gov} \rangle, \ldots \}\}
Relational algebra (RA)

• An algebraic query language for querying relational data based on “operators”
• Not used in commercial DBMSs
  • use “real” language SQL
• But SQL basically implements (extended) RA
• SQL query gets translated into RA for optimizations
• RA has well-defined meaning – builds the foundation of database queries in SQL
Relational algebra Operators

- **Core operators:**
  - Selection, projection, cross product, union, difference, and renaming
- **Additional, derived operators:**
  - Join, natural join, intersection, etc.
- **Compose operators to make complex queries**
Selection

• Input: a table $R$

• Notation: $\sigma_p R$
  • $p$ is called a selection condition (or predicate)

• Purpose: filter rows according to some criteria

• Output: same columns as $R$, but only rows or $R$ that satisfy $p$
Selection example

• Users with popularity higher than 0.5

\[ \sigma_{\text{pop}>0.5} \text{User} \]

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>0.3</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

\[ \sigma_{\text{pop}>0.5} \]
More on selection

- Selection condition can include any column of $R$, constants, comparison ($=$, $\leq$, etc.) and Boolean connectives ($\land$: and, $\lor$: or, $\neg$: not)
  - Example: users with popularity at least 0.9 and age under 10 or above 12

$$\sigma(?????) \; User$$
More on selection

• Selection condition can include any column of $R$, constants, comparison ($=$, $\leq$, etc.) and Boolean connectives ($\land$: and, $\lor$: or, $\neg$: not)
  • Example: users with popularity at least 0.9 and age under 10 or above 12
    
    \[ \sigma_{pop \geq 0.9 \land (age < 10 \lor age > 12)} \text{ User} \]

• You must be able to evaluate the condition over each single row of the input table!
  • Example: the most popular user
    
    \[ \sigma_{pop \geq \text{every pop in User}} \text{ User} \]

WRONG!
Projection

• Input: a table $R$
• Notation: $\pi_L R$
  • $L$ is a list of columns in $R$
• Purpose: output chosen columns
• Output: same rows, but only the columns in $L$
Projection example

- IDs and names of all users

\[ \pi_{uid, name} \text{User} \]

<table>
<thead>
<tr>
<th>$uid$</th>
<th>$name$</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>0.3</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>
More on projection

• Duplicate output rows are removed (by definition)
  • Example: user ages

\[ \pi_{age} User \]

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>142</td>
<td>Bart</td>
<td>10</td>
<td>0.9</td>
</tr>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>456</td>
<td>Ralph</td>
<td>8</td>
<td>0.3</td>
</tr>
<tr>
<td>…</td>
<td>…</td>
<td>…</td>
<td>…</td>
</tr>
</tbody>
</table>

\[ \pi_{age} \]
Cross product

• Input: two tables \( R \) and \( S \)
• Natation: \( R \times S \)
• Purpose: pairs rows from two tables
• Output: for each row \( r \) in \( R \) and each \( s \) in \( S \), output a row \( rs \) (concatenation of \( r \) and \( s \))
## Cross product example

### User×Member

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
<td>123</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
<td>857</td>
<td>abc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>857</td>
<td>gov</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The cross product combines elements from two tables: `User` and `Member`.
- The resulting table has a unique `uid` for each combination.
- The `gid` column indicates the group identifier.

### Table 1: User

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Table 2: Member

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>abc</td>
</tr>
<tr>
<td>857</td>
<td>gov</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- The cross product ensures each user is matched with each member, resulting in a comprehensive list.

---

*Note: The image contains a visual representation of the cross product with nodes and arrows indicating the relationship between the tables.*
A note a column ordering

- Recall: Ordering of columns is unimportant as far as contents are concerned

\[
\begin{array}{c|c|c|c|c}
\text{uid} & \text{name} & \text{age} & \text{pop} & \text{gid} \\
123 & Milhouse & 10 & 0.2 & 123 \\
123 & Milhouse & 10 & 0.2 & 857 \\
123 & Milhouse & 10 & 0.2 & 857 \\
857 & Lisa & 8 & 0.7 & 123 \\
857 & Lisa & 8 & 0.7 & 857 \\
857 & Lisa & 8 & 0.7 & 857 \\
\ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\quad = \quad
\begin{array}{c|c|c|c|c}
\text{uid} & \text{gid} & \text{uid} & \text{name} & \text{age} & \text{pop} \\
123 & gov & 123 & Milhouse & 10 & 0.2 \\
857 & abc & 123 & Milhouse & 10 & 0.2 \\
857 & gov & 123 & Milhouse & 10 & 0.2 \\
123 & gov & 857 & Lisa & 8 & 0.7 \\
857 & abc & 857 & Lisa & 8 & 0.7 \\
857 & gov & 857 & Lisa & 8 & 0.7 \\
\ldots & \ldots & \ldots & \ldots & \ldots & \ldots \\
\end{array}
\]

- So cross product is **commutative**, i.e., for any \( R \) and \( S \), \( R \times S = S \times R \) (up to the ordering of columns)
Derived operator: join

(A.k.a. “theta-join”)

• Input: two tables $R$ and $S$

• Notation: $R \bowtie_p S$
  • $p$ is called a join condition (or predicate)

• Purpose: relate rows from two tables according to some criteria

• Output: for each row $r$ in $R$ and each row $s$ in $S$, output a row $rs$ if $r$ and $s$ satisfy $p$

• Shorthand for $\sigma_p (R \times S)$

• An important operator with various scope for optimization!
Join example

- Info about users, plus IDs of their groups

$\text{User} \bowtie_{\text{User.uid} = \text{Member.uid}} \text{Member}$

Prefix a column reference with table name and “.” to disambiguate identically named columns from different tables.
Derived operator: natural join

• Input: two tables $R$ and $S$
• Notation: $R \bowtie S$
• Purpose: relate rows from two tables, and
  • Enforce equality between identically named columns
  • Eliminate one copy of identically named columns
• Shorthand for $\pi_L(R \bowtie \rho S)$, where
  • $\rho$ equates each pair of columns common to $R$ and $S$
  • $L$ is the union of column names from $R$ and $S$ (with duplicate columns removed)
Natural join example

\[
\text{User} \bowtie \text{Member} = \pi_? (\text{User} \bowtie_? \text{Member})
\]

User (uid, name, age, pop)
Member (uid, gid)
Natural join example

User \Join Member = \pi_? (User \Join Member)
= \pi_{uid, name, age, pop, gid} (User \Join User.uid= Member.uid)

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uid</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>abc</td>
</tr>
<tr>
<td>857</td>
<td>gov</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
<td>abc</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
<td>gov</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Natural join example

$User \bowtie Member = \pi_{uid,name,age,pop,gid}(User \bowtie_{uid=Member.uid} Member)$

<table>
<thead>
<tr>
<th>uid</th>
<th>name</th>
<th>age</th>
<th>pop</th>
<th>gid</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Milhouse</td>
<td>10</td>
<td>0.2</td>
<td>gov</td>
</tr>
<tr>
<td>857</td>
<td>Lisa</td>
<td>8</td>
<td>0.7</td>
<td>abc</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>gov</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
</tbody>
</table>

$User \bowtie Member$
Union

• Input: two tables $R$ and $S$
• Notation: $R \cup S$
  • $R$ and $S$ must have identical schema
• Output:
  • Has the same schema as $R$ and $S$
  • Contains all rows in $R$ and all rows in $S$ (with duplicate rows removed)
Difference

• Input: two tables $R$ and $S$
• Notation: $R - S$
  • $R$ and $S$ must have identical schema
• Output:
  • Has the same schema as $R$ and $S$
  • Contains all rows in $R$ that are not in $S$
Derived operator: intersection

• Input: two tables $R$ and $S$
• Notation: $R \cap S$
  • $R$ and $S$ must have identical schema
• Output:
  • Has the same schema as $R$ and $S$
  • Contains all rows that are in both $R$ and $S$
• Q. How do you write intersection using the previous operators?
Derived operator: intersection

- Input: two tables $R$ and $S$
- Notation: $R \cap S$
  - $R$ and $S$ must have identical schema
- Output:
  - Has the same schema as $R$ and $S$
  - Contains all rows that are in both $R$ and $S$
- Q. How do you write intersection using the previous operators?
  - Shorthand for $R - (R - S)$
  - Also equivalent to $S - (S - R)$
  - And to $R \bowtie S$
Renaming

• Input: a table $R$ and $S$
• Notation: $\rho_S R$, $\rho_{(A_1, A_2, \ldots)} R$, or $\rho_{S(A_1, A_2, \ldots)} R$
• Purpose: “rename” a table and/or its columns
• Output: a table with the same rows as $R$, but called differently
• Used to
  • Avoid confusion caused by identical column names
  • Create identical column names for natural joins
• As with all other relational operators, it doesn’t modify the database
  • Think of the renamed table as a copy of the original
Renaming example

• IDs of users who belong to at least two groups
  \[ \text{Member} \Join \exists ? \text{Member} \]

\[
\pi_{\text{uid}} \left( \text{Member} \Join \text{Member.uid=Member.uid} \land \text{Member.gid\neq Member.gid} \right)
\]

WRONG!

\[
\pi_{\text{uid}_1} \left( \rho_{(\text{uid}_1,\text{gid}_1)} \text{Member} \Join \exists \text{uid}_1=\text{uid}_2 \land \text{gid}_1\neq\text{gid}_2 \right)
\]

\[
\rho_{(\text{uid}_2,\text{gid}_2)} \text{Member}
\]
"Expression tree" or "Logical Query Plan Tree" notation

User (uid, name, age, pop)
Member (uid, gid)

Q. What does this RA expression output?
Summary of core operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Union: $R \cup S$
- Difference: $R - S$
- Renaming: $\rho_S(A_1, A_2, \ldots) R$
  - Does not really add “processing” power
Summary of derived operators

• Join: $R \bowtie_p S$
• Natural join: $R \bowtie S$
• Intersection: $R \cap S$

• Many more
  • Semijoin, anti-semijoin, quotient, ...
An exercise

• Names of users in Lisa’s groups

Writing a query bottom-up:
An exercise

• Names of users in Lisa’s groups

Writing a query bottom-up:

Who’s Lisa?

σ_{name="Lisa"} User → Member → Member

Lisa’s groups

π_{gid} Member → User

Users in Lisa’s groups

π_{uid} User

Their names

π_{name}
Another exercise

• IDs of groups that Lisa doesn’t belong to

Writing a query top-down:
Another exercise

• IDs of groups that Lisa doesn’t belong to

Writing a query top-down:

\[
\begin{align*}
\pi_{gid} \quad &\quad \pi_{gid} \\
\text{Member} \quad &\quad \text{Member} \\
\end{align*}
\]

\[
\begin{align*}
\sigma_{name=\text{"Lisa"}} \\
\text{User} \\
\end{align*}
\]

User (uid, name, age, pop)
Member (uid, gid)
A trickier exercise

• Who are the most popular?

User (uid, name, age, pop)
Member (uid, gid)
A trickier exercise

• Who are the most popular?
  • Who do NOT have the highest pop rating?
  • Whose pop is lower than somebody else’s?

A deeper question:
When (and why) is “—” needed?
Monotone operators

• If some old output rows may need to be removed
  • Then the operator is non-monotone

• Otherwise the operator is monotone
  • That is, old output rows always remain “correct” when more rows are added to the input

• Formally, for a monotone operator $\text{op}$:
  $R \subseteq R'$ implies $\text{op}(R) \subseteq \text{op}(R')$ for any $R, R'$
Classification of relational operators

- Selection: $\sigma_p R$
- Projection: $\pi_L R$
- Cross product: $R \times S$
- Join: $R \bowtie_p S$
- Natural join: $R \bowtie S$
- Union: $R \cup S$
- Difference: $R - S$
- Intersection: $R \cap S$

Monotone or not?
Classification of relational operators

• Selection: $\sigma_p R$  Monotone
• Projection: $\pi_L R$  Monotone
• Cross product: $R \times S$  Monotone
• Join: $R \bowtie_p S$  Monotone
• Natural join: $R \bowtie S$  Monotone
• Union: $R \cup S$  Monotone
• Difference: $R - S$  Monotone w.r.t. $R$; non-monotone w.r.t $S$
• Intersection: $R \cap S$  Monotone
Why is “−” needed for “highest”?

• Composition of monotone operators produces a monotone query
  • Old output rows remain “correct” when more rows are added to the input

• Is the “highest” query monotone?
  • No!
  • Current highest pop is 0.9
  • Add another row with pop 0.91
  • Old answer is invalidated

☞ So it must use difference!
Why do we need core operator $X$?

• Difference
  • The only non-monotone operator

• Projection
  • The only operator that removes columns

• Cross product
  • The only operator that adds columns

• Union
  • The only operator that allows you to add rows?
  • A more rigorous argument?

• Selection?
  • Homework problem
Extensions to relational algebra

• Duplicate handling (“bag algebra”)
• Grouping and aggregation
• “Extension” (or “extended projection”) to allow new column values to be computed

➦ All these will come up when we talk about SQL
➦ But for now we will stick to standard relational algebra without these extensions
Why is RA a good query language?

• Simple
  • A small set of core operators
  • Semantics are easy to grasp

• **Declarative?**
  • Declares “what” is needed, not “how”
  • Yes, compared with older languages like CODASYL
  • Though operators do look somewhat “procedural”

• Complete?
  • With respect to what?
  • SQL is “relationally-complete” = it is at least as expressive as RA = every RA expression has an equivalent expression in SQL
Relational calculus (First order logic for databases)

• Introduced by Codd along with Rel. model and algebra

\{u.uid \mid u \in User \land \\
\neg (\exists u' \in User: u.pop < u'.pop)\}, or

\{u.uid \mid u \in User \land \\
(\forall u' \in User: u.pop \geq u'.pop)\}

• Relational algebra = “safe” relational calculus
  • Every query expressible as a safe relational calculus query is also expressible as a relational algebra query
  • And vice versa

• Example of an “unsafe” relational calculus query
  • \{u.name \mid \neg (u \in User)\}
  • Not “domain-independent, i.e. Cannot evaluate it just by looking at the database
Turing machine

• A conceptual device that can execute any computer algorithm
• Approximates what general-purpose programming languages can do
  • E.g., Python, Java, C++, …

☞ So how does relational algebra compare with a Turing machine?
Limits of relational algebra

• Relational algebra has no recursion
  • Example: given relation $\text{Friend}(\text{uid1}, \text{uid2})$, who can Bart reach in his social network with any number of hops?
    • Writing this query in RA is impossible!
    • So RA is not as powerful as general-purpose languages

• But why not?
  • Optimization becomes undecidable
  <$\Rightarrow$> Simplicity is empowering
  • Besides, you can always implement it at the application level, and recursion is added to SQL nevertheless!
Announcements (Wed, Jan 18)
Announcements : 1/3 (Wed. Jan 18)

• Sign up for Piazza & Gradiance
• Change in midterm date: 02/22 (W) (from 02/20 (M))
  • To have enough time to go through solutions of HW2
• Homework #1 assigned
  • Due on 02/06
  • Start solving problems soon after the topics are covered
• Set up VM (instructions on course website)
• Next Wednesday: Yuhao and Junyang will walk through and help with VM setup for those who need it
  • Start solving the problems on paper and later try on VM
• The gradiance assignments will be posted “after” the corresponding lectures (by 8 pm)
Announcements : 2/3 (Wed. Jan 18)

• Reminder: Homework Policy

• You need to solve the problems on your own
• You can discuss ideas with your classmates, but mention names and acknowledge all helps you have received
• You cannot copy solution from another student
• Solutions must come from “your head”
  • i.e. you have to “own” the solution
• You cannot “search for” solutions from online material, forums, previous years’ assignments, or any other sources
• Any violation will have serious consequences
• If in doubt whether something is allowed, send me (the instructor) an email and ask
Announcements: 3/3 (Wed. Jan 18)

• Project ideas and requirements to be posted by next class
• You do not have to start forming groups right away
• There will be a “project mixer” class (after two weeks)
  • If you have an idea, give a pitch preparing a few slides
  • If you like an idea, join a group with space
  • There will be a few rounds of random reshuffling of your seats after the presentations, so you will meet your classmates and discuss ideas
  • Look for project partners with diverse expertise and ideas

• Expected group size = 4
  • If 3, still have to do the same work
  • Do not go below 3
  • Only if there is no exact division by 4, may go to group size of 5 (at the end, not before)